

APPENDIX A

```
#include "optionact4.hpp"
#include <iostream.h>
#include <stdio.h>

int main(void)
{
    int
        NombrePeriodes=500,
        NPI = NombrePeriodes/2,
        Epaisseur=2,
        AjusteTheta = 0;

    double
        Frequence=0.5,
        Maturite=10.0,
        MatI =5.0,
        Prix,
        P=2.0/3.0,
        A=0.58,
        B=0.0345,
        AngleTheta = 23.0*3.141592654/75.0,
        *DeltaT,
        Strike=0.0,
        OrigineAction=40,
        Correlation=-0.5,
        VolatiliteAction=0.25,
        VolatiliteTaux=0.03,
        Delta[2],
        Gamma[3],
        Theta,
        OrigineTaux=0.08,
        Coupon=0.025,
        Ratio=2.0,
        Nominal=100.0,
        Rappel=105,
        SMax = 700,
        SMin=1,
        RMax = 0.30;

    flux=fopen("e:\\test.txt", "a");
    //for(SoftCalls[0].Trigger = 80; SoftCalls[0].Trigger <= 130; SoftCalls[0].Trigger += 4)
    {
        //for(Correlation=-0.5; Correlation <= 0.5; Correlation += 0.1)
        //for(P=0.5; P <= 0.66; P += 0.01)
        //for(AngleTheta=0.0*3.141592654/3.0; AngleTheta <= 1.0*3.141592654/3.0;
    AngleTheta += 3.141592654/30.0)
        //for(NombrePeriodes = 100; NombrePeriodes <= 1000; NombrePeriodes+=100)
        {
            DeltaT = new double [NombrePeriodes + 1];
            NPI=NombrePeriodes/2;
            for(int I = 0; I <= NPI; I++)
                DeltaT[I]=MatI/NPI;
            for(I = NPI + 1; I <= NombrePeriodes; I++)
                DeltaT[I]=(Maturite-MatI)/(NombrePeriodes-NPI);
```

```
Prix= CalculOptionActionTauxStochastique(NombrePeriodes, DeltaT, Strike,
OrigineAction,
Correlation, VolatiliteAction, VolatiliteTaux, P, A, B, Delta,
Gamma, &Theta, OrigineTaux, Coupon, Ratio, Frequence,
Nominal,Rappel, SMax, SMin,RMax, AngleTheta,
AjusteTheta);

cout << NombrePeriodes << endl;
cout << Prix << endl;

delete DeltaT;
fprintf(flux,"%f\n", Prix);
}

/*Prix= CalculOptionActionTauxStochastique(NombrePeriodes, Epaisseur,DeltaT, Strike,
OrigineAction,
Correlation, VolatiliteAction, VolatiliteTaux, P, A, B, Delta,
Gamma, &Theta, OrigineTaux, Coupon, Ratio, Frequence, Nominal,Rappel);*/
// fprintf( stream , " % f \ t % f \ t % f \ t % f \ t % f \ t % f \ n " ,
Prix,Delta[0],Delta[1],Gamma[0],Gamma[1],Gamma[2]);
}
fclose(flux);

return 1;
}
```

// OPTIONACT4.HPP

//-----

```
#ifndef OPTIONACTIONTAUXSTOCHASTIQUE_HPP
#define OPTIONACTIONTAUXSTOCHASTIQUE_HPP
```

```
#include "taux.hpp"
#include "matrice.hpp"
#include "systemes.hpp"
#include <math.h>
#include <stdio.h>
#include <iostream.h>
#include <time.h>
#include <stdlib.h>
#include <memory.h>
```

```
#define Position(X, Y) (Y) * K3 + (X) + K2
```

```
FILE
```

```
*flux;
```

```
class CCoordonnee
```

```
{
public:
    int
        X,
        Y;
};
```

//-----

```
class CTranche
```

```
{
public:
    CCoordonnee
```

```
        *Noeuds; // Tous les noeuds de la tranche
double
        *Prix;
int
        XMin,
        XMax,
        YMin,
        YMax,
        *BordGauche,
        *BordDroit;

CTranche(void) {};
~CTranche(void) { delete [] BordGauche; delete [] BordDroit; };
};
```

```
class COptionActionTauxStochastique
{
public:
    CCoordonnee
        *AllNoeudsP,
        *AllNoeudsF;
    double
        *PrixP,
        *PrixF;
    CTranche
        *Tranches;
    double
        DUX,
        DUY,
        CC,
        Transformation[2][2],
        Transformation2[2][2],
        Transformation20[2][2],
        Probabilites[7],
        RacineTrois,
        RacineOrigineTaux,
        DeuxRacineOrigineTaux,
        VolActionCarre,
        VolTauxCarre,
        VolActionCarreSurDeux,
        VolTauxCarreSurDeux,
        NextAction,
        NextTaux,
        *ATheta,
        *sqrtDeltaT;
    int
        K2,
        K3,
        Compteur,
        GrandLargeurMax;

    // Parametres
    int
        AjusteTheta,
        NombrePeriodes;
    double
        Prix,
        Frequence,
        A,
        B,
        P,
```

```

    AngleTheta,
    Coupon,
    Ratio,
    Nominal,
    *DeltaT,
    *Temps,
    VolatiliteAction,
    VolatiliteTaux,
    Strike,
    Correlation,
    OrigineAction,
    OrigineTaux,
    Rappel,
    SMax,
    SMin,
    RMax;

void Init(void);
void Construire(void);
void ConstruireTranche(int IndicePeriode);
void ConstruireFils(CCoordonnee Noeud, CCoordonnee Fils[7], int IndicePeriode, double Drift[2],
    double Transformation20[2][2], double Transformation2[2][2]);
void CalculTransformation(double Transformation2[2][2], int IndicePeriode);
void CalculActionTaux(int IndicePeriode, int X, int Y, double *Action, double *Taux, double
T[2][2]);
void Liberer(void);

void ChercherNoeud(int IndicePeriode, CCoordonnee Noeud[7], CCoordonnee *NoeudFils[7]);

void CalculPayOff(void);
void CalculDescente(int IndicePeriode);
void Parcourir(void);

void CalculProbabilites(int IndicePeriode, CCoordonnee Noeud, CCoordonnee Fils[7], double
Drift[2]);
double PrixOC(int IndicePeriode, double Action);
double PrixOC2(int IndicePeriode, double Prix, double Action);
};

//-----

void COptionActionTauxStochastique::Init(void)
{
    RacineTrois = sqrt(3.0);
    RacineOrigineTaux = sqrt(OrigineTaux);
    DeuxRacineOrigineTaux = 2.0 * RacineOrigineTaux;
    VolActionCarre = VolatiliteAction * VolatiliteAction;
    VolTauxCarre = VolatiliteTaux * VolatiliteTaux;
    VolActionCarreSurDeux = VolActionCarre / 2.0;
    VolTauxCarreSurDeux = VolTauxCarre / 2.0;
    CC = sqrt(A * A + 2 * VolTauxCarre);
    ATheta = new double [NombrePeriodes + 1];
    for(int I = 0; I <= NombrePeriodes; I++)
        ATheta[I] = 0.17856527 / ((double) rand() / (double) RAND_MAX * 3.141592654 / 3.0);
    GrandLargeurMax = 0;
    Temps = new double [NombrePeriodes + 1];
    Temps[0] = 0.0;
    for(I = 1; I <= NombrePeriodes; I++)
        Temps[I] = Temps[I - 1] + DeltaT[I];
    sqrtDeltaT = new double [NombrePeriodes + 1];
    for(I = 0; I <= NombrePeriodes; I++)

```

```
sqrtDeltaT[I] = sqrt(DeltaT[I]);

CalculTransformation(Transformation2, 0);
double
    n,
    nmax = 0.0,
    P[4][2],
    P2[4][2],
    MinDT = 1.0e20;

for(I = 0; I <= NombrePeriodes; I++)
    if(MinDT > DeltaT[I])
        MinDT = DeltaT[I];
P[0][0] = log(SMin);
P[0][1] = - 2.0 * sqrt(RMax);
P[1][0] = log(SMin);
P[1][1] = 2.0 * sqrt(RMax);
P[2][0] = log(SMax);
P[2][1] = 2.0 * sqrt(RMax);
P[3][0] = log(SMax);
P[3][1] = - 2.0 * sqrt(RMax);
for(I = 0; I < 4; I++)
{
    P[I][0] = log(OrigineAction);
    P[I][1] = 2.0 * sqrt(OrigineTaux);
    SystemeLineaire2Inconnues(Transformation2[0][0], Transformation2[0][1], P[I][0],
        Transformation2[1][0], Transformation2[1][1], P[I][1], &P2[I][0], &P2[I][1]);
    P2[I][0] /= sqrt(MinDT);
    P2[I][1] /= sqrt(MinDT);
    n = sqrt(P2[I][0] * P2[I][0] + P2[I][1] * P2[I][1]);
    if(nmax < n)
        nmax = n;
}
GrandLargeurMax = Round(nmax) + 10;
AllNoeudsP = new CCoordonnee [(2 * GrandLargeurMax + 1) * (2 * GrandLargeurMax + 1)];
AllNoeudsF = new CCoordonnee [(2 * GrandLargeurMax + 1) * (2 * GrandLargeurMax + 1)];
PrixP = new double [(2 * GrandLargeurMax + 1) * (2 * GrandLargeurMax + 1)];
PrixF = new double [(2 * GrandLargeurMax + 1) * (2 * GrandLargeurMax + 1)];
K2 = 2 * GrandLargeurMax * (1 + GrandLargeurMax);
K3 = 2 * GrandLargeurMax + 1;

cout << "Taille " << 2 * (sizeof(int)+sizeof(double)) * (2 * GrandLargeurMax + 1) * (2 *
GrandLargeurMax + 1) << endl;
}

void COptionActionTauxStochastique::Construire(void)
{
    int
        I,
        J,
        IndicePeriode;
    CCoordonnee
        *Noeuds;
    double
        *PrixTemp;

    Tranches = new CTranche [NombrePeriodes + 1];

    memset(PrixP, 127, sizeof(double) * (2 * GrandLargeurMax + 1) * (2 * GrandLargeurMax + 1));
    Tranches[0].Noeuds = AllNoeudsP;
    Tranches[0].Prix = PrixP;
```

```

Tranches[0].BordGauche = new int [2 * GrandLargeurMax + 1];
Tranches[0].BordDroit = new int [2 * GrandLargeurMax + 1];
Tranches[0].BordGauche[GrandLargeurMax] = 0;
Tranches[0].BordDroit[GrandLargeurMax] = 0;
Tranches[0].Prix[Position(0, 0)] = - 1.0;
Tranches[0].XMin = Tranches[0].XMax =      Tranches[0].YMin = Tranches[0].YMax = 0;

for(IndicePeriode = 1; IndicePeriode <= NombrePeriodes; IndicePeriode++)
{
    //cout << IndicePeriode << endl;
    memset(PrixF, 127, sizeof(double) * (2 * GrandLargeurMax + 1) * (2 *
GrandLargeurMax + 1));
    Tranches[IndicePeriode].Noeuds = AllNoeudsF;
    Tranches[IndicePeriode].Prix = PrixF;
    ConstruireTranche(IndicePeriode);
    memcpy(Transformation20, Transformation2, sizeof(double) * 4);
    Noeuds = AllNoeudsP;
    AllNoeudsP = AllNoeudsF;
    AllNoeudsF = Noeuds;
    Tranches[IndicePeriode].Noeuds = AllNoeudsP;
    PrixTemp = PrixP;
    PrixP = PrixF;
    PrixF = PrixTemp;
    Tranches[IndicePeriode].Prix = PrixP;
}
}

```

```

void COptionActionTauxStochastique::ConstruireTranche(int IndicePeriode)
{

```

```

    CCoordonnee
        Node,
        Fils[7];
    int
        I,
        J,
        K,
        Pos,
        XX,
        YY,
        XMin = 2 << 29,
        XMax = - XMin,
        YMin = 2 << 29,
        YMax = - YMin;
    double
        Drift[2];
    int
        MaxN,
        GrandMaxN = 0,
        LP;
    double
        Action,
        Taux,
        BonTheta,
        ThetaTemp,
        ThetaMin=0.0,
        ThetaMax=3.141592654/3.0,
        DeltaTheta=ThetaMax/5.0;
    double
        *Tempo;

    if(AjusteTheta)

```

```

{
    if(IndicePeriode<10)
        BonTheta=23.0*3.141592/75.0;
    else
    {
        for(ThetaTemp = ThetaMin; ThetaTemp <= ThetaMax; ThetaTemp +=
DeltaTheta)
        {
            ATheta[IndicePeriode-1]=ThetaTemp;
            CalculTransformation(Transformation2, IndicePeriode-1);
            T e m p o = n e w d o u b l e
            [(2*GrandLargeurMax+1)*(2*GrandLargeurMax+1)];
            memset(Tempo, 127, 8 * (2 * GrandLargeurMax + 1) * (2 *
GrandLargeurMax + 1));
            MaxN = 0;
            for(J = Tranches[IndicePeriode - 1].YMin; J <=
Tranches[IndicePeriode - 1].YMax; J++)
                for(I = Tranches[IndicePeriode - 1].BordGauche[J +
GrandLargeurMax];
                    I <= Tranches[IndicePeriode - 1].BordDroit[J +
GrandLargeurMax]; I+=10)
                {
                    Pos = Position(I, J);
                    if(Tranches[IndicePeriode - 1].Prix[Pos] < 0.0)
                    {
                        Node.X = I;
                        Node.Y = J;
                        ConstruireFils(Node, Fils,
IndicePeriode - 1, Drift, Transformation20, Transformation2);
                        for(K = 6; K >= 0; K--)
                        {
                            Pos = Position(Fils[K].X,
Fils[K].Y);
                            if(Tempo[Pos] > 0.0)
                            {
                                Tempo[Pos]=-1.0;
                                MaxN++;
                            }
                        }
                    }
                }
            delete Tempo;
            if(GrandMaxN < MaxN)
            {
                GrandMaxN = MaxN;
                BonTheta=ThetaTemp;
            }
            cout << MaxN << " ";
        }
        cout << " / " << GrandMaxN << " " << BonTheta << endl;
    }
    ATheta[IndicePeriode-1]=BonTheta;
}
CalculTransformation(Transformation2, IndicePeriode - 1);

Tranches[IndicePeriode].BordGauche = new int [2 * GrandLargeurMax + 1];
Tranches[IndicePeriode].BordDroit = new int [2 * GrandLargeurMax + 1];
memset(Tranches[IndicePeriode].BordGauche, 127, sizeof(int) * (2 * GrandLargeurMax + 1));
memset(Tranches[IndicePeriode].BordDroit, 250, sizeof(int) * (2 * GrandLargeurMax + 1));
for(J = Tranches[IndicePeriode - 1].YMin; J <= Tranches[IndicePeriode - 1].YMax; J++)
    for(I = Tranches[IndicePeriode - 1].BordGauche[J + GrandLargeurMax];

```

```

        I <= Tranches[IndicePeriode - 1].BordDroit[J + GrandLargeurMax]; I++)
    {
        Pos = Position(I, J);
        if(Tranches[IndicePeriode - 1].Prix[Pos] < 0.0)
        {
            Node.X = I;
            Node.Y = J;
            ConstruireFils(Node, Fils, IndicePeriode - 1, Drift, Transformation20,
Transformation2);
            for(K = 6; K >= 0; K--)
            {
                CalculActionTaux(IndicePeriode, XX = Fils[K].X, YY =
Fils[K].Y, &Action, &Taux, Transformation2);
                if((Action > SMin) && (Action < SMax) && (Taux <
RMax))
                {
                    Pos = Position(XX, YY);
                    if(Tranches[IndicePeriode].Prix[Pos] > 0.0)
                    {
                        i f ( X X <
Tranches[IndicePeriode].BordGauche[YY + GrandLargeurMax])
Tranches[IndicePeriode].BordGauche[YY + GrandLargeurMax] = XX;
                        i f ( X X >
Tranches[IndicePeriode].BordDroit[YY + GrandLargeurMax])
Tranches[IndicePeriode].BordDroit[YY + GrandLargeurMax] = XX;
                        Tranches[IndicePeriode].Prix[Pos] =
- 1.0;
                        if(XX < XMin)
                            XMin = XX;
                        if(XX > XMax)
                            XMax = XX;
                        if(YY < YMin)
                            YMin = YY;
                        if(YY > YMax)
                            YMax = YY;
                    }
                }
            }
        }
        Tranches[IndicePeriode].XMin = XMin;
        Tranches[IndicePeriode].XMax = XMax;
        Tranches[IndicePeriode].YMin = YMin;
        Tranches[IndicePeriode].YMax = YMax;

        /*FILE *stream = fopen("f:\\test.txt", "a");
        for(I = 0; I < Tranches[IndicePeriode].NombreNoeudsContour; I++)
            f p r i n t f ( s t r e a m , " % d \ t % d \ n " ,
Tranches[IndicePeriode].Contour[I].X, Tranches[IndicePeriode].Contour[I].Y);
        fprintf(stream, "\n");
        fclose(stream);*/
    }

inline void COptionActionTauxStochastique::ConstruireFils(CCoordonnee Noeud, CCoordonnee Fils[7],
int IndicePeriode, double Drift[2], double Transformation20[2][2], double Transformation2[2][2])
{
    double
        Y,
        iX,

```



```
iY,
XX,
YY,
U, V,
max,
s,
Coins[2][4],
RacineT = sqrtDeltaT[IndicePeriode],
RacineT0 = IndicePeriode ? sqrtDeltaT[IndicePeriode - 1] : 0.0;

int
    I,
    II;

if(IndicePeriode == 0)
    Y = DeuxRacineOrigineTaux;
else
    Y = (Transformation20[1][0] * Noeud.X + Transformation20[1][1] * Noeud.Y) *
RacineT0 +
    DeuxRacineOrigineTaux;
Drift[0] = (Y * Y / 4.0 - VolActionCarreSurDeux) * DeltaT[IndicePeriode];
Drift[1] = (2.0 * A * (B - Y * Y / 4.0) / Y - VolTauxCarreSurDeux / Y) *
    DeltaT[IndicePeriode];
SystemeLineaire2Inconnues(Transformation[0][0], Transformation[0][1], Drift[0],
    Transformation[1][0], Transformation[1][1], Drift[1], &XX, &YY);
if(IndicePeriode == 0)
{
    U = XX / RacineT;
    V = YY / RacineT;
    XX = (XX - YY / RacineTrois) / RacineT;
    YY = (2.0 * YY / RacineTrois) / RacineT;
}
else
{
    double
        An = ATheta[IndicePeriode - 1] - ATheta[IndicePeriode],
        CosAn = cos(An),
        SinAn = sin(An);
    U = ((Noeud.X * CosAn + (CosAn - SinAn * RacineTrois) / 2.0 * Noeud.Y) * RacineT0
+ XX) / RacineT;
    V = ((Noeud.X * SinAn + (SinAn + CosAn * RacineTrois) / 2.0 * Noeud.Y) * RacineT0
+ YY) / RacineT;
    XX = (U - V / RacineTrois);
    YY = (2.0 * V / RacineTrois);
}
iX = floor(XX);
iY = floor(YY);
Coins[0][0] = iX + iY / 2.0;
Coins[1][0] = Coins[1][1] = iY * RacineTrois / 2.0;
Coins[1][2] = Coins[1][3] = Coins[1][0] + RacineTrois / 2.0;
Coins[0][1] = Coins[0][0] + 1.0;
Coins[0][2] = iX + (iY + 1) / 2.0;
Coins[0][3] = Coins[0][2] + 1.0;
max = 1.0e30;
for(I = 3; I >= 0; I--)
{
    s = (Coins[0][I] - U) * (Coins[0][I] - U) + (Coins[1][I] - V) * (Coins[1][I] - V);
    if(s < max)
    {
        max = s;
        II = I;
    }
}
```

```

    }
    Fils[6].X = Fils[1].X = Fils[4].X = (int) iX + (II % 2);
    Fils[6].Y = Fils[0].Y = Fils[3].Y = (int) iY + (II >= 2);
    Fils[0].X = Fils[5].X = Fils[6].X + 1;
    Fils[2].X = Fils[3].X = Fils[6].X - 1;
    Fils[1].Y = Fils[2].Y = Fils[6].Y + 1;
    Fils[4].Y = Fils[5].Y = Fils[6].Y - 1;
}

void COptionActionTauxStochastique::CalculTransformation(double Transformation2[2][2], int
IndicePeriode)
{
    double
        A11 = VolActionCarre,
        A22 = VolTauxCarre,
        A12 = Correlation * VolatiliteAction * VolatiliteTaux,
        ValeursPropres[2],
        VecteursPropres[2][2],
        Temp;

    if(Correlation == 0.0)
    {
        ValeursPropres[0] = A11;
        ValeursPropres[1] = A22;
        VecteursPropres[0][0] = 1.0;
        VecteursPropres[0][1] = 0.0;
        VecteursPropres[1][0] = 0.0;
        VecteursPropres[1][1] = 1.0;
    }
    else
    {
        ValeursPropres[0] = (A11 + A22 + sqrt((A11 - A22) * (A11 - A22) + 4.0 * A12 * A12))
/ 2.0;
        ValeursPropres[1] = (A11 + A22 - sqrt((A11 - A22) * (A11 - A22) + 4.0 * A12 * A12))
/ 2.0;

        if(fabs(ValeursPropres[0] - ValeursPropres[1]) < 1.0e-10)
        {
            VecteursPropres[0][0] = 1.0;
            VecteursPropres[0][1] = 0.0;
            VecteursPropres[1][0] = 0.0;
            VecteursPropres[1][1] = 1.0;
        }
        else
        {
            VecteursPropres[0][0] = - A12;
            VecteursPropres[0][1] = A11 - ValeursPropres[0];
            VecteursPropres[1][0] = VecteursPropres[0][1];
            VecteursPropres[1][1] = A12;
        }
    }

    Temp = sqrt(VecteursPropres[0][0] * VecteursPropres[0][0] + VecteursPropres[0][1] *
VecteursPropres[0][1]);
    VecteursPropres[0][0] /= Temp;
    VecteursPropres[0][1] /= Temp;
    Temp = sqrt(VecteursPropres[1][0] * VecteursPropres[1][0] + VecteursPropres[1][1] *
VecteursPropres[1][1]);
    VecteursPropres[1][0] /= Temp;
    VecteursPropres[1][1] /= Temp;
    ValeursPropres[0] = sqrt(ValeursPropres[0]);
    ValeursPropres[1] = sqrt(ValeursPropres[1]);
}

```

```
AngleTheta = ATheta[IndicePeriode];
double
    CosAngleTheta = cos(AngleTheta),
    SinAngleTheta = sin(AngleTheta),
    Sqrt2P = sqrt(2.0 / P);
/*Transformation[0][0] = (CosAngleTheta * VecteursPropres[0][0] * ValeursPropres[0] +
    SinAngleTheta * VecteursPropres[0][1] * ValeursPropres[1]) * Sqrt2P;
Transformation[0][1] = (- SinAngleTheta * VecteursPropres[0][0] * ValeursPropres[0] +
    CosAngleTheta * VecteursPropres[0][1] * ValeursPropres[1]) * Sqrt2P;
Transformation[1][0] = (CosAngleTheta * VecteursPropres[1][0] * ValeursPropres[0] +
    SinAngleTheta * VecteursPropres[1][1] * ValeursPropres[1]) * Sqrt2P;
Transformation[1][1] = (- SinAngleTheta * VecteursPropres[1][0] * ValeursPropres[0] +
    CosAngleTheta * VecteursPropres[1][1] * ValeursPropres[1]) * Sqrt2P;*/

Transformation[0][0] = ((CosAngleTheta * VecteursPropres[0][0] + SinAngleTheta *
VecteursPropres[1][0])
    * VecteursPropres[0][0] * ValeursPropres[0] +
    (SinAngleTheta * VecteursPropres[1][1] +
    CosAngleTheta * VecteursPropres[0][1]) * VecteursPropres[0][1] * ValeursPropres[1])
    * Sqrt2P;
Transformation[0][1] = ((CosAngleTheta * VecteursPropres[1][0] - SinAngleTheta *
VecteursPropres[0][0])
    * VecteursPropres[0][0] * ValeursPropres[0] +
    (-SinAngleTheta * VecteursPropres[0][1] +
    CosAngleTheta * VecteursPropres[1][1]) * VecteursPropres[0][1] * ValeursPropres[1])
    * Sqrt2P;
Transformation[1][0] = ((CosAngleTheta * VecteursPropres[0][0] + SinAngleTheta *
VecteursPropres[1][0])
    * VecteursPropres[1][0] * ValeursPropres[0] +
    (SinAngleTheta * VecteursPropres[1][1] +
    CosAngleTheta * VecteursPropres[0][1]) * VecteursPropres[1][1] * ValeursPropres[1])
    * Sqrt2P;
Transformation[1][1] = ((CosAngleTheta * VecteursPropres[1][0] - SinAngleTheta *
VecteursPropres[0][0])
    * VecteursPropres[1][0] * ValeursPropres[0] +
    (-SinAngleTheta * VecteursPropres[0][1] +
    CosAngleTheta * VecteursPropres[1][1]) * VecteursPropres[1][1] * ValeursPropres[1])
    * Sqrt2P;

Transformation2[0][0] = Transformation[0][0];
Transformation2[1][0] = Transformation[1][0];
Transformation2[0][1] = (Transformation[0][0] + Transformation[0][1] * RacineTrois) / 2.0;
Transformation2[1][1] = (Transformation[1][0] + Transformation[1][1] * RacineTrois) / 2.0;
}

inline void COptionActionTauxStochastique::CalculActionTaux(int IndicePeriode, int X, int Y, double
*Action,
    double *Taux, double T[2][2])
{
    double
        DT = IndicePeriode ? sqrtDeltaT[IndicePeriode - 1] : 0.0,
        Ta = (T[1][0] * X + T[1][1] * Y) * DT + DeuxRacineOrigineTaux;

    *Taux = Ta * Ta / 4.0;
    *Action = exp((T[0][0] * X + T[0][1] * Y) * DT) * OrigineAction;
}

void COptionActionTauxStochastique::Liberer(void)
{
    delete [] AllNoeudsP;
    delete [] AllNoeudsF;
}
```

```

delete PrixP;
delete PrixF;
delete [] Tranches;
delete Temps;
delete ATheta;
delete sqrtDeltaT;
}

inline void COptionActionTauxStochastique::ChercherNoeud(int IndicePeriode, CCoordonnee Noeud[7],
CCoordonnee *NoeudFils[7])
{
    int
        I;

    for(I = 6; I >= 0; I--)
        NoeudFils[I] = Tranches[IndicePeriode + 1].Noeuds + Position(Noeud[I].X, Noeud[I].Y);
}

void COptionActionTauxStochastique::CalculPayOff(void)
{
    int
        I,
        J,
        IndicePeriode = NombrePeriodes,
        Pos;

    double
        Action,
        Taux;

    CCoordonnee
        *Noeuds = Tranches[IndicePeriode].Noeuds;

    for(J = Tranches[IndicePeriode].YMin; J <= Tranches[IndicePeriode].YMax; J++)
        for(I = Tranches[IndicePeriode].BordGauche[J + GrandLargeurMax];
            I <= Tranches[IndicePeriode].BordDroit[J + GrandLargeurMax]; I++)
        {
            Pos = Position(I, J);
            if(Tranches[IndicePeriode].Prix[Pos] < 0.0)
            {
                CalculActionTaux(NombrePeriodes, I, J, &Action, &Taux,
Transformation20);
                Noeuds[Pos].X = I;
                Noeuds[Pos].Y = J;
                if((Action < SMax) && (Taux < RMax))
                    Tranches[IndicePeriode].Prix[Pos] = PrixOC(IndicePeriode,
Action);
                else if(Action >= SMax)
                    Tranches[IndicePeriode].Prix[Pos] = Ratio * SMax;
                else
                    Tranches[IndicePeriode].Prix[Pos] = Ratio * Action;
            }
        }
}

void COptionActionTauxStochastique::CalculDescente(int IndicePeriode)
{
    int
        I,
        J,
        K,
        Pos,
        Pos2,

```

```

Ok,
NombreNoeuds = (2 * GrandLargeurMax + 1) * (2 * GrandLargeurMax + 1);
CCoordonnee
Fils[7];
CCoordonnee
*NoeudFils[7];
double
Prix,
Action,
Taux,
Drift[2];

for(J = Tranches[IndicePeriode].YMin; J <= Tranches[IndicePeriode].YMax; J++)
for(I = Tranches[IndicePeriode].BordGauche[J + GrandLargeurMax];
I <= Tranches[IndicePeriode].BordDroit[J + GrandLargeurMax]; I++)
{
Pos = Position(I, J);
Tranches[IndicePeriode].Noeuds[Pos].X = I;
Tranches[IndicePeriode].Noeuds[Pos].Y = J;
CalculActionTaux(IndicePeriode, I, J, &Action, &Taux, Transformation20);
if((Action < SMax) && (Taux < RMax))
{
ConstruireFils(Tranches[IndicePeriode].Noeuds[Pos], Fils,
IndicePeriode, Drift,
Transformation20, Transformation2);
Ok = 1;
for(K = 6; K >= 0; K--)
{
Pos2 = Position(Fils[K].X, Fils[K].Y);
if((Pos2 < 0) || (Pos2 >= NombreNoeuds))
{
Ok = 0;
Tranches[IndicePeriode].Prix[Pos] = 1.0e30;
break;
}
}
if(Ok)
{
ChercherNoeud(IndicePeriode, Fils, NoeudFils);
CalculProbabilites(IndicePeriode,
Tranches[IndicePeriode].Noeuds[Pos], Fils, Drift);
Prix = 0.0;
for(K = 6; K >= 0; K--)
{
int
PosP = Position(Fils[K].X, Fils[K].Y);
if(Tranches[IndicePeriode + 1].Prix[PosP] >
1.0e25)
{
CalculActionTaux(IndicePeriode + 1,
Fils[K].X, Fils[K].Y, &Action, &Taux, Transformation2);
if(Action >= SMax)
Tranches[IndicePeriode +
1].Prix[PosP] = Ratio * SMax;
else if((Taux >= RMax) && (Action
> SMin))
Tranches[IndicePeriode +
1].Prix[PosP] = Ratio * Action;
else if((Taux < RMax) && (Action <
SMin))
{

```

```

double
    Ob,
    AA,
    BB,
    C = 0.0;

    AA = pow(2.0 * CC *
exp((CC + A) * (Temps[NombrePeriodes] - Temps[IndicePeriode]) / 2.0) /
    ((CC + A) *
    (exp(CC * (Temps[NombrePeriodes] - Temps[IndicePeriode])) - 1.0) + CC + CC),
    2.0 * A * B /
VolTauxCarre);
    BB = 2.0 * (exp(CC *
    (Temps[NombrePeriodes] - Temps[IndicePeriode])) - 1.0) /
    ((CC + A) *
    (exp(CC * (Temps[NombrePeriodes] - Temps[IndicePeriode])) - 1.0) + CC + CC);
    Ob = (1.0 + Coupon) * AA
    * exp(- BB * Taux);
    while (C <=
    Temps[NombrePeriodes]
    Temps[IndicePeriode])
    {
        if (C >
        {
            AA =
            pow(2.0 * CC * exp((CC + A) * (C - Temps[IndicePeriode]) / 2.0) /
            ((CC + A) * (exp(CC * (C - Temps[IndicePeriode])) - 1.0) + CC + CC),
            2.0 * A * B / VolTauxCarre);
            BB=20
            * (exp(CC * (C - Temps[IndicePeriode])) - 1.0) /
            ((CC + A) * (exp(CC * (C - Temps[IndicePeriode])) - 1.0) + CC + CC);
            Ob +=
            Coupon * AA * exp(- BB * Taux);
        }
        C += Frequence;
    }
    Tranches[IndicePeriode +
1].Prix[PosP] = Ob * Nominal;
    }
    else
    {
        Tranches[IndicePeriode].Prix[Pos] = 1.0e30;
        break;
    }
    Prix += Probabilites[K] * Tranches[IndicePeriode
+ 1].Prix[PosP];
    }
    if(K < 0)
    {
        Prix *= exp(- Taux * DeltaT[IndicePeriode]);
        Tranches[IndicePeriode].Prix[Pos] =
        PrixOC2(IndicePeriode, Prix, Action);
        /*ConstruireFils(Tranches[IndicePeriode].Noeuds[Pos],
        Transformation20, Transformation2);

```

```

double
    s1=0.0,s2=0.0,s3=0.0,s4=0.0,s5=0.0,
    xp = (Transformation20[0][0] *
Tranches[IndicePeriode].Noeuds[Pos].X +
Transformation20[0][1] *
Tranches[IndicePeriode].Noeuds[Pos].Y * sqrt(DeltaT[IndicePeriode-1]),
    yp = (Transformation20[1][0] *
Tranches[IndicePeriode].Noeuds[Pos].X +
Transformation20[1][1] *
Tranches[IndicePeriode].Noeuds[Pos].Y * sqrt(DeltaT[IndicePeriode-1]));
    for(int Z = 0; Z < 7; Z++)
    {
        double
            xi=(Transformation2[0][0]
* Fils[Z].X +
Transformation2[0][1] * Fils[Z].Y * sqrt(DeltaT[IndicePeriode]),
            yi=(Transformation2[1][0]
* Fils[Z].X +
Transformation2[1][1] * Fils[Z].Y * sqrt(DeltaT[IndicePeriode]));
        s1 += Probabilites[Z] *
        s2 += Probabilites[Z] *
        s3 += Probabilites[Z] *
        s4 += Probabilites[Z] *
        s5 += Probabilites[Z] *
        (xi-xp);
        (yi-yp);
        pow(xi-xp,2);
        pow(yi-yp,2);
        (xi-xp)*(yi-yp);
        }
        s1 = Drift[0];
        s2 = Drift[1];
        s3 = Drift[2];
        s4 = Drift[3];
        s5 = Drift[4];
        pow(VolatiliteAction,2)*DeltaT[IndicePeriode]+pow(Drift[0],2);
        pow(VolatiliteTaux,2)*DeltaT[IndicePeriode]+pow(Drift[1],2);
        VolatiliteTaux*VolatiliteAction*DeltaT[IndicePeriode]+Drift[0]*Drift[1];
        ss=abs(s1)+abs(s2)+abs(s3)+abs(s4)+abs(s5);
        if(ss > 1.0e-14)
            cout << "ss" << endl;*/
    }
}
else if(Action >= SMax)
    Tranches[IndicePeriode].Prix[Pos] = Ratio * SMax;
else
    Tranches[IndicePeriode].Prix[Pos] = Ratio * Action;
}
}

void COptionActionTauxStochastique::Parcours(void)
{
    int
        I,
        IndicePeriode;
    CCoordonnee
        *Noeuds;

```

```

double
    *PrixTemp;

CalculPayOff();
for(IndicePeriode = NombrePeriodes - 1; IndicePeriode > 0; IndicePeriode--)
{
    //cout << IndicePeriode << endl;
    Noeuds = AllNoeudsP;
    AllNoeudsP = AllNoeudsF;
    AllNoeudsF = Noeuds;
    PrixTemp = PrixP;
    PrixP = PrixF;
    PrixF = PrixTemp;
    memset(PrixP, 127, sizeof(double) * (2 * GrandLargeurMax + 1) * (2 *
GrandLargeurMax + 1));
    Tranches[IndicePeriode + 1].Noeuds = AllNoeudsF;
    Tranches[IndicePeriode].Noeuds = AllNoeudsP;
    Tranches[IndicePeriode + 1].Prix = PrixF;
    Tranches[IndicePeriode].Prix = PrixP;
    CalculTransformation(Transformation20, IndicePeriode - 1);
    CalculTransformation(Transformation2, IndicePeriode);
    CalculDescente(IndicePeriode);
}
Noeuds = AllNoeudsP;
AllNoeudsP = AllNoeudsF;
AllNoeudsF = Noeuds;
PrixTemp = PrixP;
PrixP = PrixF;
PrixF = PrixTemp;
memset(PrixP, 127, sizeof(double) * (2 * GrandLargeurMax + 1) * (2 * GrandLargeurMax + 1));
Tranches[1].Noeuds = AllNoeudsF;
Tranches[0].Noeuds = AllNoeudsP;
Tranches[1].Prix = PrixF;
Tranches[0].Prix = PrixP;
CalculTransformation(Transformation2, 0);
CalculDescente(0);
Prix = Tranches[0].Prix[Position(0, 0)];
}

void COptionActionTauxStochastique::CalculProbabilites(int IndicePeriode, CCoordonnee Noeud,
CCoordonnee Fils[7], double Drift[2])
{
    double
        An,
        X,
        Y,
        Alpha,
        Beta,
        RacineT = sqrt(DeltaT[IndicePeriode]),
        SinAn,
        CosAn;

    if(IndicePeriode == 0)
    {
        X = Fils[6].X * RacineT;
        Y = Fils[6].Y * RacineT;
    }
    else
    {
        An = ATheta[IndicePeriode - 1] - ATheta[IndicePeriode];
        SinAn = sin(An);
    }
}

```



```
CosAn = cos(An);
X = Fils[6].X * RacineT - (Noeud.X * (CosAn - SinAn / RacineTrois) -
    2.0 * SinAn / RacineTrois * Noeud.Y) * sqrtDeltaT[IndicePeriode - 1];
Y = Fils[6].Y * RacineT - (2.0 * SinAn / RacineTrois * Noeud.X +
    (CosAn + SinAn / RacineTrois) * Noeud.Y) * sqrtDeltaT[IndicePeriode - 1];
}
Drift[0] -= Transformation2[0][0] * X + Transformation2[0][1] * Y;
Drift[1] -= Transformation2[1][0] * X + Transformation2[1][1] * Y;
SystemeLineaire2Inconnues(Transformation[0][0], Transformation[0][1], Drift[0],
    Transformation[1][0], Transformation[1][1], Drift[1], &Alpha, &Beta);
Alpha /= RacineT;
Beta /= RacineT;

double
    Alpha3 = Alpha / 3.0,
    Alpha2 = Alpha * Alpha,
    Beta2 = Beta * Beta,
    BetaR3 = Beta / RacineTrois;
Probabilites[6] = 1.0 - P - Alpha2 - Beta2;
Probabilites[0] = (P - Beta2) / 6.0 + Alpha2 / 2.0 + Alpha3;
Probabilites[3] = Probabilites[0] - Alpha3 - Alpha3;
Probabilites[1] = (P + Alpha) / 6.0 + Beta2 / 3.0 + Alpha * BetaR3 + BetaR3 / 2.0;
Probabilites[4] = Probabilites[1] - Alpha3 - BetaR3;
Probabilites[2] = Probabilites[1] - Alpha3 - 2.0 * Alpha * BetaR3;
Probabilites[5] = Probabilites[2] + Alpha3 - BetaR3;
/*for(int K = 0; K < 7; K++)
    if(Probabilites[K] < -1.0e-16)
        cout << "<0" << endl;*/
}

inline double COptionActionTauxStochastique::PrixOC(int IndicePeriode, double Action)
{
    double
        Result = (1.0 + Coupon) * Nominal;
    /*if(Result > Rappel)
        Result = Rappel;*/
    if(Result < Ratio * Action)
        Result = Ratio * Action;
    if(Result < Strike)
        Result = Strike;
    return Result;
}

inline double COptionActionTauxStochastique::PrixOC2(int IndicePeriode, double Prix, double Action)
{
    double
        Result = Prix,
        T = Temps[IndicePeriode] / Frequence;

    if(IndicePeriode && (fabs(Round(T) - T) < 1.0e-12))
        Result += Coupon * Nominal;
    if(Result < Ratio * Action)
        Result = Ratio * Action;
    if(Result > Rappel)
        Result = Rappel;
    if(Result < Strike)
        Result = Strike;
    return Result;
}

//-----
```

```
double CalculOptionActionTauxStochastique(int NombrePeriodes, double *DeltaT,
double Strike, double OrigineAction,
double Correlation, double VolatiliteAction, double VolatiliteTaux, double P,
double A, double B, double Delta[2],
double Gamma[3], double *Theta, double OrigineTaux, double Coupon, double Ratio, double
Frequence,
double Nominal, double Rappel, double SMax, double SMin, double RMax, double AngleTheta,
int AjusteTheta)
{
    COptionActionTauxStochastique
        Modele;
    double
        Prix;

    Modele.NombrePeriodes = NombrePeriodes;
    Modele.DeltaT = DeltaT;
    Modele.Strike = Strike;
    Modele.VolatiliteAction = VolatiliteAction;
    Modele.VolatiliteTaux = VolatiliteTaux;
    Modele.Correlation = Correlation;
    Modele.OrigineAction = OrigineAction;
    Modele.OrigineTaux = OrigineTaux;
    Modele.P = P;
    Modele.A = A;
    Modele.B = B;
    Modele.Coupon=Coupon;
    Modele.Ratio=Ratio;
    Modele.Frequence=Frequence;
    Modele.Nominal=Nominal;
    Modele.Rappel=Rappel;
    Modele.SMax = SMax;
    Modele.SMin = SMin;
    Modele.RMax = RMax;
    Modele.AngleTheta = AngleTheta;
    Modele.AjusteTheta = AjusteTheta;

    clock_t
        t0,
        t1,
        t2;

    Modele.Init();
    cout << "Construction" << endl;
    t0 = clock();
    Modele.Construire();
    t1 = clock();
    cout << "Parcours" << endl;
    Modele.Parcours();
    t2 = clock();
    cout << (double) (t1 - t0) / CLOCKS_PER_SEC << " secondes" << endl;
    cout << (double) (t2 - t1) / CLOCKS_PER_SEC << " secondes" << endl;
    Prix = Modele.Prix;

    Modele.Liberer();

    return Prix;
}

#endif
```